case_study

August 13, 2024

[17]:
```
import pandas as pd

pricing_data = pd.read_csv("Competition_Data.csv")
```

[18]:
```
pricing_data.head()
```

[18]:

	Index	Fiscal_Week_ID	Store_ID	Item_ID	Price	Item_Quantity	Sales_Amount_No_Discount	Sales_Amount	Competition_Price
0	0	2019-11	store_459	item_526	134.49	435	4716.74	11272.59	206.44
1	1	2019-11	store_459	item_526	134.49	435	4716.74	11272.59	158.01
2	2	2019-11	store_459	item_526	134.49	435	4716.74	11272.59	278.03
3	3	2019-11	store_459	item_526	134.49	435	4716.74	11272.59	222.66
4	4	2019-11	store_459	item_526	134.49	435	4716.74	11272.59	195.32

[19]:
```
pricing_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Index                    100000 non-null  int64
 1   Fiscal_Week_ID           100000 non-null  object
 2   Store_ID                 100000 non-null  object
 3   Item_ID                  100000 non-null  object
 4   Price                    100000 non-null  float64
 5   Item_Quantity            100000 non-null  int64
 6   Sales_Amount_No_Discount 100000 non-null  float64
 7   Sales_Amount             100000 non-null  float64
 8   Competition_Price        100000 non-null  float64
dtypes: float64(4), int64(2), object(3)
memory usage: 6.9+ MB
```

[20]:
```
import matplotlib.pyplot as plt
```

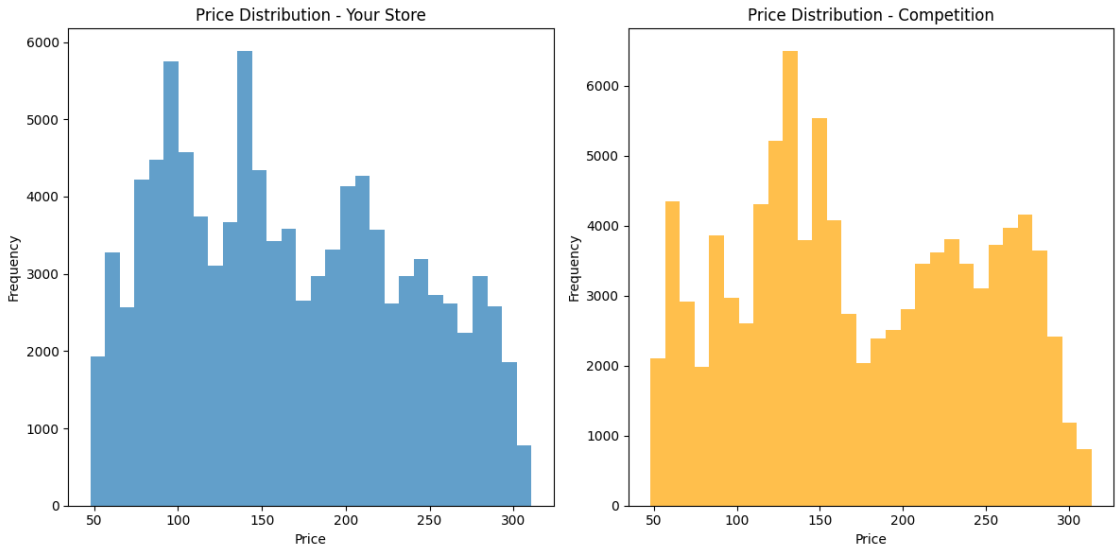
0.0.1 let's start by comparing the price distribution with the competition:

[21]:
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(pricing_data['Price'], bins=30, alpha=0.7, label='Your Store')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price Distribution - Your Store')

plt.subplot(1, 2, 2)
plt.hist(pricing_data['Competition_Price'], bins=30, alpha=0.7, color='orange',␣
↪label='Competition')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Price Distribution - Competition')

plt.tight_layout()
plt.show()
```



1) It shows that the competition's prices are generally higher, with peaks around the 100-150 and 200-250 price ranges, which indicate a concentration of higher-priced items. In contrast, our store's prices are more evenly distributed across the 50-300 range, with notable peaks around 100-150.

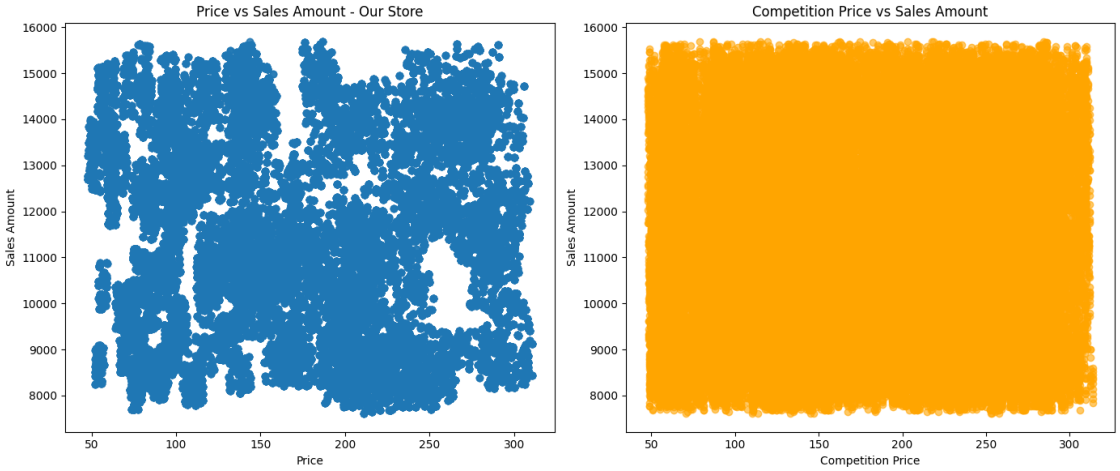
0.0.2 let's compare the relationship between price and sales:

[22]:
```
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(pricing_data['Price'], pricing_data['Sales_Amount'], alpha=0.6,␣
 ↪label='Your Store')
plt.xlabel('Price')
plt.ylabel('Sales Amount')
plt.title('Price vs Sales Amount - Our Store')

plt.subplot(1, 2, 2)
plt.scatter(pricing_data['Competition_Price'], pricing_data['Sales_Amount'],␣
 ↪alpha=0.6, color='orange', label='Competition')
plt.xlabel('Competition Price')
plt.ylabel('Sales Amount')
plt.title('Competition Price vs Sales Amount')

plt.tight_layout()
plt.show()
```



1) The scatter plots compare the relationship between price and sales amount for our store (left) and the competition (right). For our store, the plot shows a wide dispersion of sales amounts across various price points, which indicates varied performance in different price ranges without a clear trend. In contrast, the competition's plot shows a dense clustering of sales amounts around higher values, with prices also spread across a similar range but demonstrating a more consistent sales performance. It suggests that the competition might have a more effective pricing strategy, which maintains higher sales amounts more uniformly across different price points.

0.0.3 let's compare the price changes over time:

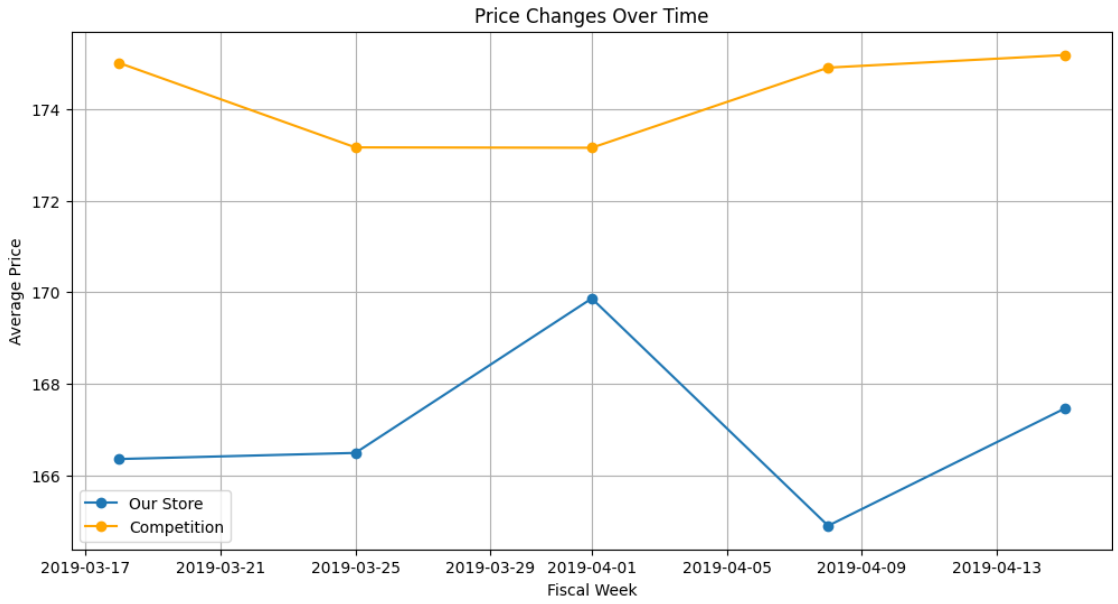
[23]:
```
pricing_data['Fiscal_Week_ID'] = pd.to_datetime(pricing_data['Fiscal_Week_ID']␣
 ↪+ '-1', format='%Y-%U-%w')

weekly_prices = pricing_data.groupby('Fiscal_Week_ID').agg({
    'Price': 'mean',
    'Competition_Price': 'mean'
}).reset_index()

plt.figure(figsize=(12, 6))

plt.plot(weekly_prices['Fiscal_Week_ID'], weekly_prices['Price'], label='Our␣
 ↪Store', marker='o')
plt.plot(weekly_prices['Fiscal_Week_ID'], weekly_prices['Competition_Price'],␣
 ↪label='Competition', marker='o', color='orange')

plt.xlabel('Fiscal Week')
plt.ylabel('Average Price')
plt.title('Price Changes Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



1) The competition maintains higher average prices consistently above 170, with a slight upward trend over the observed weeks. In contrast, our store's prices start around 166, increase slightly, then dip before rising again. It indicates that the competition follows a more stable pricing strategy, while our store experiences more fluctuations in pricing. The stability in

the competition's pricing could be contributing to their higher and more consistent sales performance.

0.0.4 let's analyze how changes in prices affect the change in quantity sold. For this, we need to calculate price elasticity. Here's the formula used to calculate price elasticity:

2) Ed = % change in quantity demanded / % change in price

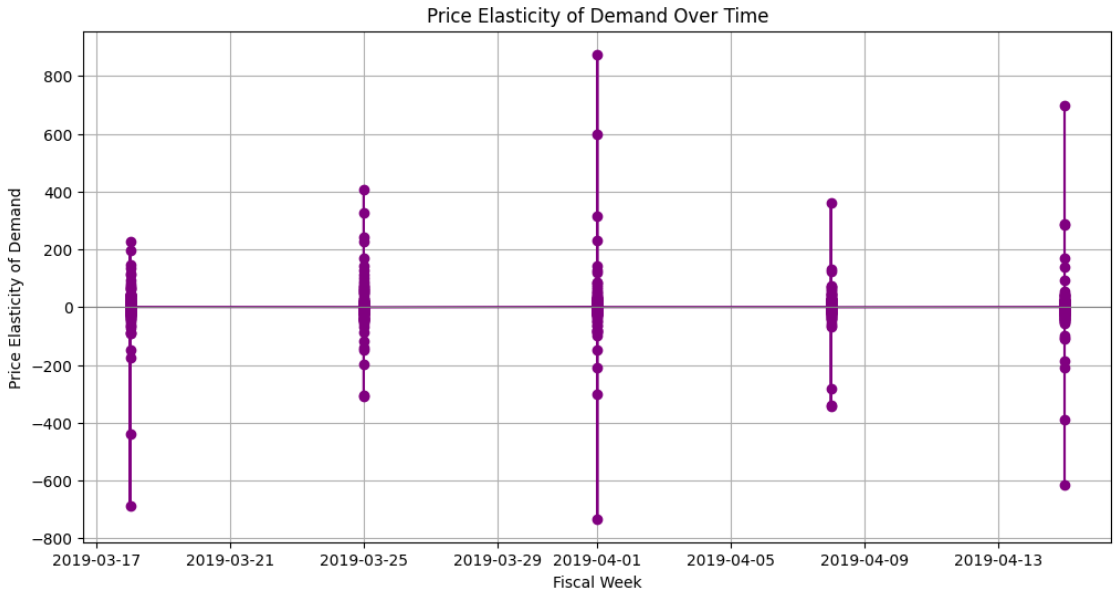
Let's calculate and visualize the price elasticity:

[24]:
```
pricing_data['price_change'] = pricing_data['Price'].pct_change()
pricing_data['qty_change'] = pricing_data['Item_Quantity'].pct_change()

pricing_data['elasticity'] = pricing_data['qty_change'] /␣
 ↪pricing_data['price_change']

pricing_data.replace([float('inf'), -float('inf')], float('nan'), inplace=True)
pricing_data.dropna(subset=['elasticity'], inplace=True)

plt.figure(figsize=(12, 6))
plt.plot(pricing_data['Fiscal_Week_ID'], pricing_data['elasticity'],␣
 ↪marker='o', linestyle='-', color='purple')
plt.axhline(0, color='grey', linewidth=0.8)
plt.xlabel('Fiscal Week')
plt.ylabel('Price Elasticity of Demand')
plt.title('Price Elasticity of Demand Over Time')
plt.grid(True)
plt.show()
```



1) The graph shows the price elasticity of demand over time. It highlights significant variability in elasticity across different weeks, with values ranging from highly negative to highly positive. It indicates that the sensitivity of quantity demanded to price changes fluctuates considerably. High positive elasticity suggests that in some weeks, demand increased significantly with price increases, while high negative elasticity in other weeks indicates a sharp drop in demand with price hikes. The broad spread of elasticity values implies an inconsistent response to price changes, which suggests that factors other than price, such as promotions, seasonality, or market conditions, might be influencing demand.

0.0.5 let's calculate and compare the total sales amounts for our store and the competition:

[25]:
```
total_sales_your_store = pricing_data['Sales_Amount'].sum()
total_sales_competition = (pricing_data['Competition_Price'] *␣
 ↪pricing_data['Item_Quantity']).sum()

total_qty_your_store = pricing_data['Item_Quantity'].sum()
total_qty_competition = pricing_data['Item_Quantity'].sum()  # assuming␣
 ↪quantities sold are the same for comparison

summary = pd.DataFrame({
    'Metric': ['Total Sales Amount', 'Total Quantity Sold'],
    'Your Store': [total_sales_your_store, total_qty_your_store],
    'Competition': [total_sales_competition, total_qty_competition]
})


summary
```

[25]:

	Metric	Your Store	Competition
0	Total Sales Amount	1.141005e+08	6.962097e+08
1	Total Quantity Sold	3.984776e+06	3.984776e+06

1) Our store's total sales amount is 114,100,500, whereas the competition's total sales amount is 696,209,700 (assuming equal quantity sold). The competition has a significantly higher total sales amount compared to our store. It indicates that their pricing strategy is more effective in generating revenue.

0.0.6 Now, we'll analyze how the sales amounts vary across different price brackets to identify if there are specific price ranges where the competition outperforms our store:

[26]:
```
# define price brackets
bins = [0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500]
labels = ['0-50', '51-100', '101-150', '151-200', '201-250', '251-300',␣
 ↪'301-350', '351-400', '401-450', '451-500']
```

```
# create price brackets for both your store and competition
pricing_data['price_bracket'] = pd.cut(pricing_data['Price'], bins=bins,␣
 ↪labels=labels, right=False)
pricing_data['competition_price_bracket'] = pd.
 ↪cut(pricing_data['Competition_Price'], bins=bins, labels=labels, right=False)

# calculate sales amount by price bracket for your store
sales_by_bracket_your_store = pricing_data.
 ↪groupby('price_bracket')['Sales_Amount'].sum().reset_index()
sales_by_bracket_your_store.columns = ['Price Bracket', 'Your Store Sales␣
 ↪Amount']

# calculate sales amount by price bracket for competition
pricing_data['competition_sales_amt'] = pricing_data['Competition_Price'] *␣
 ↪pricing_data['Item_Quantity']
sales_by_bracket_competition = pricing_data.
 ↪groupby('competition_price_bracket')['competition_sales_amt'].sum().
 ↪reset_index()
sales_by_bracket_competition.columns = ['Price Bracket', 'Competition Sales␣
 ↪Amount']

sales_by_bracket = pd.merge(sales_by_bracket_your_store,␣
 ↪sales_by_bracket_competition, on='Price Bracket')

sales_by_bracket
```

[26]:

	Price Bracket	Your Store Sales Amount	Competition Sales Amount
0	0-50	346800.63	9.305357e+05
1	51-100	24636244.30	4.889277e+07
2	101-150	29645669.06	1.278404e+08
3	151-200	20658418.18	1.092184e+08
4	201-250	20742288.10	1.909748e+08
5	251-300	16778087.66	2.047670e+08
6	301-350	1292959.36	1.358583e+07
7	351-400	0.00	0.000000e+00
8	401-450	0.00	0.000000e+00
9	451-500	0.00	0.000000e+00

The table shows the total sales amounts for both our store and the competition across different price brackets. Here are some key observations:

1) 0-50 Bracket: The competition has significantly higher sales in this bracket.
2) 51-100 Bracket: The competition outperforms our store by a wide margin.
3) 101-150 Bracket: The competition's sales are much higher than our store's sales.
4) 151-200 Bracket: The competition again has significantly higher sales.
5) 201-250 Bracket: The competition's sales are nearly double those of our store.

6) 251-300 Bracket: The competition has higher sales, but the gap is smaller compared to other brackets.
7) 301-350 Bracket: The competition has higher sales, though the overall sales amount is lower in this bracket compared to others.

0.0.7 Price Optimization with Dynamic Pricing

Let's start by defining a dynamic pricing model and simulating its performance. Here are the steps :

1) We will enhance our dataset to include segments and calculate price elasticity for each segment.
2) We'll create segments based on purchasing behaviour and calculate price elasticity for each segment.
3) We'll define dynamic pricing rules based on competitor pricing, demand, and elasticity.
4) We'll simulate the dynamic pricing model and compare it with the existing pricing strategy.

0.0.8 let's start with segmenting the data and calculating price elasticity for each segment:

[27]:
```
import warnings
warnings.filterwarnings('ignore')
```

[28]:
```
# segment customers based on purchasing behavior

# calculate average price and total quantity sold for each item
item_summary = pricing_data.groupby('Item_ID').agg({
    'Price': 'mean',
    'Item_Quantity': 'sum'
}).reset_index()

# merge the item summary back to the main dataset
pricing_data = pd.merge(pricing_data, item_summary, on='Item_ID', suffixes=('',␣
 ↪'_avg'))

# define segments based on average price
pricing_data['segment'] = pd.cut(pricing_data['Price_avg'], bins=[0, 50, 150,␣
 ↪300], labels=['Low', 'Medium', 'High'])

# calculate price elasticity for each segment
segments = pricing_data['segment'].unique()
elasticity_data = []

for segment in segments:
    segment_data = pricing_data[pricing_data['segment'] == segment]
    segment_data['price_change'] = segment_data['Price'].pct_change()
    segment_data['qty_change'] = segment_data['Item_Quantity'].pct_change()
```

```
    segment_data['elasticity'] = segment_data['qty_change'] /␣
 ↪segment_data['price_change']
    segment_data.replace([float('inf'), -float('inf')], float('nan'),␣
 ↪inplace=True)
    avg_elasticity = segment_data['elasticity'].mean()
    elasticity_data.append({'segment': segment, 'avg_elasticity':␣
 ↪avg_elasticity})

elasticity_df = pd.DataFrame(elasticity_data)

elasticity_df
```

[28]:

	segment	avg_elasticity
0	Medium	0.154444
1	High	0.148043

1) Medium Segment: An average elasticity of 0.154444. This indicates that the quantity demanded is more elastic to price changes.
2) High Segment: An average elasticity of 0.148043. This indicates that the quantity demanded is relatively inelastic to price changes.

1) In the above code, we are segmenting customers based on their purchasing behaviour by analyzing the average price and total quantity sold for each item. First, we calculated the average price and total quantity sold for each item and merged this summary back into the main dataset. We then defined customer segments based on these average prices into three categories: Low, Medium, and High. For each segment, we calculated the price elasticity of demand by measuring how the percentage change in price affects the percentage change in quantity sold. Finally, we computed the average elasticity for each segment, which provides insights into how sensitive each segment is to price changes. This helps in understanding and optimizing pricing strategies for different customer segments

[29]:
```
# create a copy of the dataset for simulation
dynamic_pricing_data = pricing_data.copy()

# apply dynamic pricing rules
dynamic_pricing_data.loc[dynamic_pricing_data['segment'] == 'Medium',␣
 ↪'dynamic_price'] = dynamic_pricing_data['Price'] * 1.05
dynamic_pricing_data.loc[dynamic_pricing_data['segment'] == 'High',␣
 ↪'dynamic_price'] = dynamic_pricing_data['Price'] * 0.90

# calculate new sales amounts based on dynamic prices
dynamic_pricing_data['dynamic_sales_amt'] =␣
 ↪dynamic_pricing_data['dynamic_price'] * dynamic_pricing_data['Item_Quantity']

# compare total sales amount between existing and dynamic pricing
total_sales_existing = pricing_data['Sales_Amount'].sum()
total_sales_dynamic = dynamic_pricing_data['dynamic_sales_amt'].sum()
```

```
# compare total quantity sold between existing and dynamic pricing
total_qty_existing = pricing_data['Item_Quantity'].sum()
total_qty_dynamic = dynamic_pricing_data['Item_Quantity'].sum()  # quantity␣
 ↪sold remains the same for comparison

comparison_summary = pd.DataFrame({
    'Metric': ['Total Sales Amount', 'Total Quantity Sold'],
    'Existing Pricing': [total_sales_existing, total_qty_existing],
    'Dynamic Pricing': [total_sales_dynamic, total_qty_dynamic]
})

comparison_summary
```

[29]:

	Metric	Existing Pricing	Dynamic Pricing
0	Total Sales Amount	1.141005e+08	6.226950e+08
1	Total Quantity Sold	3.984776e+06	3.984776e+06

1) In the above code, we are simulating the impact of a dynamic pricing strategy on sales performance. First, we created a copy of the dataset for the simulation. We then applied dynamic pricing rules by increasing prices by 5% for the Medium segment and decreasing prices by 10% for the High segment. Then, we calculated new sales amounts based on these dynamic prices. Next, we compared the total sales amount and total quantity sold under the existing pricing and the dynamic pricing strategies.

1) The dynamic pricing strategy results in a significantly higher total sales amount compared to the existing pricing strategy. This indicates that the dynamic pricing approach is more effective in maximizing revenue.

[30]:
```
pricing_data['dynamic_price'] = dynamic_pricing_data['dynamic_price']
```

[31]:
```
pricing_data.head()
```

[31]:

	Index	Fiscal_Week_ID	Store_ID	Item_ID	Price	Item_Quantity	Sales_Amount_No_Discount	Sales_Amount	Competition_Price	price_change	qty_change	elasticity	price_bracket	competition_price_bracket
0	10	2019-03-18	store_709	item_526	136.79	459	4890.43	11545.08	206.44	0.017102	0.055172	3.226147	101-150	201-250
1	20	2019-03-18	store_442	item_526	138.67	458	4933.46	11517.46	206.44	0.013744				
2	30	2019-03-18	store_136	item_526	128.93	459	4962.56	11216.19	206.44	-0.070239				
3	40	2019-03-18	store_601	item_526	134.45	436	4704.89	10980.31	206.44	0.042814				
4	50	2019-03-18	store_458	item_526	134.13	435	4558.87	11168.10	206.44	-0.002380				

```
1   -0.002179   -0.158520       101-150                  201-250
2    0.002183   -0.031086       101-150                  201-250
3   -0.050109   -1.170389       101-150                  201-250
4   -0.002294    0.963661       101-150                  201-250
```

```
   competition_sales_amt   Price_avg  Item_Quantity_avg segment  dynamic_price
0               94755.96  132.061224              21792  Medium       143.6295
1               94549.52  132.061224              21792  Medium       145.6035
2               94755.96  132.061224              21792  Medium       135.3765
3               90007.84  132.061224              21792  Medium       141.1725
4               89801.40  132.061224              21792  Medium       140.8365
```

[ ]: